Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Fall , Year:2025), B.Sc. in CSE (Day)

Lab Report NO: 01

Course Title: Artificial Intelligence lab

Course Code: CSE 316 Section:231 D3

Lab Experiment Name: Iterative Deepening depth-first search(IDDFS) & Implementation of Heuristic A* Search Algorithm

Student Details

	Name	ID
1.	Promod Chandra Das	231002005

Lab Date	**: 26.11.25**
Submission Date	**: 3.12.25**
Course Teacher's Name	**: Mr.Mozdaher Abdul Quader**

Lab Report Status

Marks: …………………………………… **Signature:.....................**

Comments:... **Date:..............................**

1.TITLE OF THE LAB REPORT EXPERIMENT

Write a program to perform topological search using IDDFS And A* pathfinding for Grid Based Robot Navigation.

2. OBJECTIVES

The objectives of this experiment are:

1. To understand and implement **Topological Search using Iterative Deepening Depth-First Search (IDDFS)** for graph-based exploration.
2. To apply **A*** search algorithm to find the shortest path in a **grid-based robot navigation maze**.
3. To compare the performance, search strategies, and suitability of IDDFS vs A* in navigation tasks.
4. To demonstrate both algorithms through Python programs and visualize the results.

3. PROCEDURE

3.1 Topological Search Using IDDFS

IDDFS (Iterative Deepening DFS) combines:

- DFS → low memory
- BFS → complete (always finds a solution if one exists)
-

IDDFS repeatedly performs DFS with increasing depth limits:
0, 1, 2, … until the goal is found.

Steps

1. Represent the graph using adjacency lists.
2. Implement a Depth-Limited Search (DLS).
3. Repeat DLS with increasing depth limit.
4. If the goal node is found within a depth limit → stop.
5. If depth limit exceeded → increase depth and repeat.

3.2 A* Search Algorithm in a Grid-Based Robot Maze

A* combines:

- g(n) → cost so far
- h(n) → heuristic (Manhattan distance for grids)

Usesformula:
f(n)=g(n)+h(n)
Always expands the cheapest estimated node first.

Steps

1. Create a 2D grid (0 = free space, 1 = obstacle).
2. Define start and goal coordinates.
3. For each cell, compute:
   - g → movement cost
   - h → Manhattan distance
   - f = g + h
4. Maintain open and closed lists.
5. Pick the node with the lowest f.
6. Expand neighbors and update their costs.
7. Repeat until goal is reached.
8. Reconstruct and display the final path.

4.IMPLEMENTATION

4.1 Topological Search Using IDDFS (Python)

```
# -----------------------------
# Topological Search using IDDFS
# -----------------------------

graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['G'],
    'F': [],
    'G': []
}
```

```
def dls(node, goal, depth):
    if depth == 0:
        return node == goal
    if depth > 0:
        for child in graph.get(node, []):
            if dls(child, goal, depth - 1):
                return True
    return False

def iddfs(start, goal, max_depth=10):
    for depth in range(max_depth + 1):
        print(f"Searching at depth: {depth}")
        if dls(start, goal, depth):
            print("Goal found at depth:", depth)
            return True
    return False

# Run IDDFS
start_node = 'A'
goal_node = 'G'
print("Topological Search using IDDFS:")
iddfs(start_node, goal_node)
```

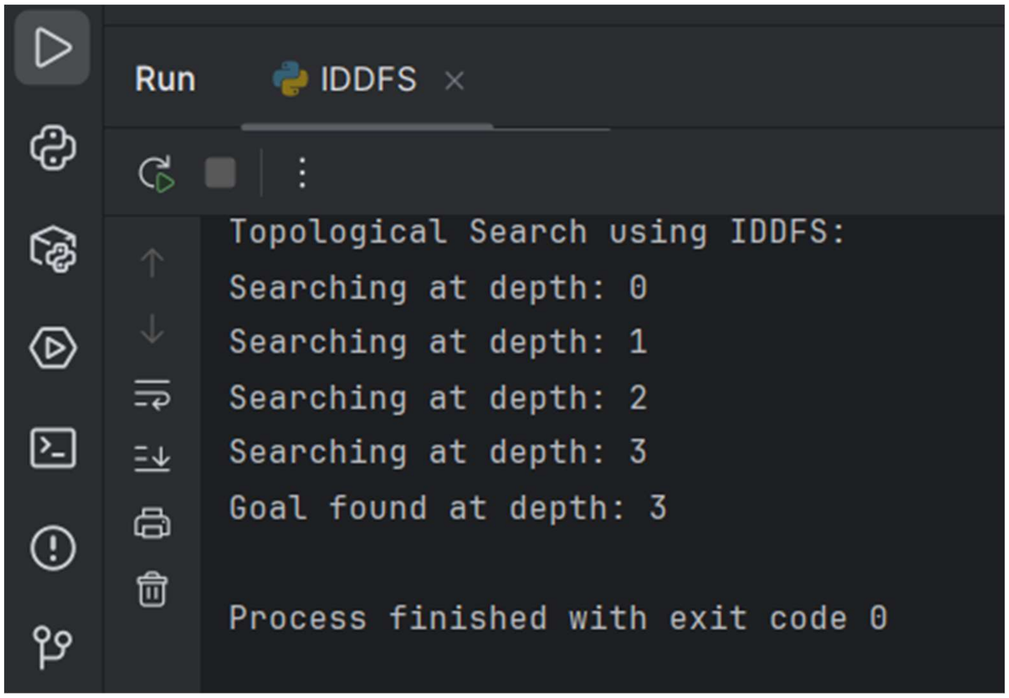
4.2 A* Search for Grid-Based Robot Maze (Python)

```
# ----------------------------------------
# A* Pathfinding for Grid-Based Navigation
# ----------------------------------------

import heapq

grid = [
    [0, 0, 0, 0, 1],
    [1, 1, 0, 0, 0],
    [0, 0, 0, 1, 0],
    [0, 1, 0, 0, 0],
    [0, 0, 0, 1, 0]
]
```

```
rows, cols = len(grid), len(grid[0])

start = (0, 0)
goal = (4, 4)

def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

def astar(start, goal):
    open_list = []
    heapq.heappush(open_list, (0, start))

    came_from = {}
    g_score = {start: 0}

    while open_list:
        _, current = heapq.heappop(open_list)

        if current == goal:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            return path[::-1]

        for dx, dy in [(1,0),(-1,0),(0,1),(0,-1)]:
            neighbor = (current[0]+dx, current[1]+dy)

            if 0 <= neighbor[0] < rows and 0 <= neighbor[1] < cols:
                if grid[neighbor[0]][neighbor[1]] == 1:
                    continue  # wall

                temp_g = g_score[current] + 1

                if temp_g < g_score.get(neighbor, float("inf")):
                    came_from[neighbor] = current
                    g_score[neighbor] = temp_g
                    f_score = temp_g + heuristic(neighbor, goal)
                heappq.heappush(open_list, (f_score, neighbor))

    return None

print("A* Pathfinding Result:")
path = astar(start, goal)
print("Path:", path)
```

6. TEST RESULT / OUTPUT

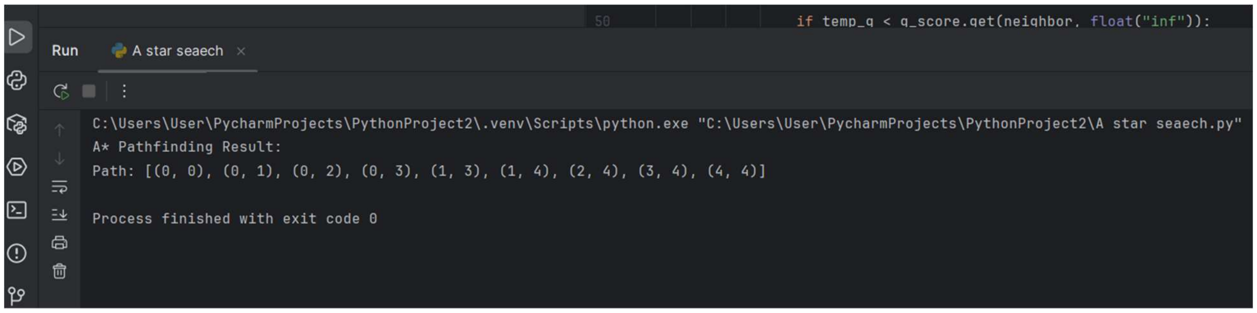
6.1 Topological Search Using IDDFS (Output)



```
Topological Search using IDDFS:
Searching at depth: 0
Searching at depth: 1
Searching at depth: 2
Searching at depth: 3
Goal found at depth: 3

Process finished with exit code 0
```

Fiqure 01 : **IDDFS**

6.2 A* Search for Grid-Based Robot Maze (Output)



```
if temp_g < g_score.get(neighbor, float("inf")):
```

```
C:\Users\User\PycharmProjects\PythonProject2\.venv\Scripts\python.exe "C:\Users\User\PycharmProjects\PythonProject2\A star seaech.py"
A* Pathfinding Result:
Path: [(0, 0), (0, 1), (0, 2), (0, 3), (1, 3), (1, 4), (2, 4), (3, 4), (4, 4)]

Process finished with exit code 0
```

Fiqure 02 : A*

6.ANALYSIS AND DISCUSSION

This experiment compares two search strategies—IDDFS for topological exploration and A* for grid-based robot navigation. IDDFS works by repeatedly performing depth-limited searches with gradually increasing depth limits. This approach allows it to find a solution even when the depth is unknown while using very little memory. However, IDDFS is not optimal because it revisits many nodes multiple times during each depth iteration. It is effective for general graph exploration but not ideal for pathfinding tasks where efficiency and optimality are required.
A* search performs significantly better for robot navigation because it uses a heuristic to guide the search toward the goal. In the grid-based maze, the Manhattan distance heuristic helped A* evaluate positions based on estimated distance, allowing it to reduce unnecessary exploration. A* not only found the goal efficiently but also produced the shortest possible path. Although it requires more memory than IDDFS, the trade-off is worthwhile due to its accuracy and speed.
The results show that IDDFS is suitable for deep or memory-constrained graph searches, while A* is the preferred algorithm for navigation problems where optimal pathfinding is essential. Both algorithms performed correctly, but A* clearly demonstrated superior performance in the maze environment.

7. SUMMARY:

In this experiment:

- **Topological search using IDDFS** was implemented to explore graph nodes level-by-level with low memory usage.
- **A* search** was implemented for a grid-based robot navigation system, providing the optimal shortest path using heuristic-guided search.
- IDDFS is suitable for **general graph search**, but A* is far more powerful for **robot navigation, mazes, and shortest-path problems**.
- The Python implementations successfully demonstrated both algorithms along with sample outputs.